



# Text Processing using Machine Learning

## Language Modelling

Liling Tan

13 Feb 2019

OVER  
**5,500** GRADUATE  
ALUMNI

OFFERING OVER  
**120** ENTERPRISE IT, INNOVATION  
& LEADERSHIP PROGRAMMES

TRAINING OVER  
**120,000** DIGITAL LEADERS  
& PROFESSIONALS

# Todays' Slides

<http://bit.ly/ANLP-Lecture5>

# Previous Notebooks...

- Lecture 1
  - <http://bit.ly/ANLP-Session1-Empty>
  - <http://bit.ly/ANLP-Session1>
- Lecture 2:
  - <http://bit.ly/ANLP-Session2-Empty->
  - <http://bit.ly/ANLP-Session2->
- Lecture 3:
  - <http://bit.ly/ANLP-Session3-Completed>
- Lecture 4:
  - <http://bit.ly/ANLP-Session4>
- Lecture 5
  - <http://bit.ly/ANLP-Session5-TextCatEmpty>
  - <http://bit.ly/ANLP-Session5NgramLM>

# Environment Setup

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration:

<https://pytorch.org/get-started/locally/>

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download <http://bit.ly/ANLP-Session5-TextCatEmpty>

Import the .ipynb to the Jupyter Notebook

## Lecture

- Language Modelling

## Hands-on

- Multi-Layered Perceptron Classification
- N-gram Language Models



# N-gram Language Modelling

# Language Model

- Language Modelling is task of ***predicting which word comes next***
- ***Predict next word  $x_i$ , given all context words up till the current word,  $x_1, \dots, x_{i-1}$***
- Language Model aka. ***assigning probability of text as an accumulated probability of all individual words given their contexts***

## Calculate Probability of a Sentence

$$P(X) = \prod_{i=1}^I P(\mathbf{x}_i | \mathbf{x}_1, \dots, \mathbf{x}_{i-1})$$

  
**Next word**      **Context**

$P(\text{"he likes to drink coffee"})$

$= P(\text{he}) + P(\text{likes} | \text{he}) + P(\text{to} | \text{he likes})$

$+ P(\text{drink} | \text{he likes to}) + P(\text{coffee} | \text{he likes to drink})$

How to compute:

$$P(X) = \prod_{i=1}^I P(\mathbf{x}_i | \mathbf{x}_1, \dots, \mathbf{x}_{i-1})$$

Count("he likes to drink coffee")  
= Count(**he**) \* Count(**likes** | **he**) \* Count(**to** | **he likes**)  
\* Count(**drink** | **he likes to**)  
\* Count(**coffee** | **he likes to drink**)

How to compute:

$$P(X) = \prod_{i=1}^I P(\mathbf{x}_i | \mathbf{x}_1, \dots, \mathbf{x}_{i-1})$$

Count("he likes to drink coffee")  
= Count(he) \* Count(likes | he) \* Count(to | he likes)  
\* Count(drink | he likes to)  
\* Count(coffee | he likes to drink)

**Too many possibilities!!!**

How to compute:

$$P(X) = \prod_{i=1}^I P(\mathbf{x}_i | \mathbf{x}_1, \dots, \mathbf{x}_{i-1})$$

$P(\text{drink} | \text{he likes to}) \approx \text{Count}(\text{drink} | \text{likes to})$

$P(\text{coffee} | \text{he likes to drink}) \approx \text{Count}(\text{coffee} | \text{to drink})$

# Ngram Language Model

## Calculate Probability of a Sentence

$$P(X) = \prod_{i=1}^I P(\mathbf{x}_i | \mathbf{x}_{i-n}, \dots, \mathbf{x}_{i-1})$$

**Next word**      **Context**  
**(specified window)**

$$\begin{aligned} & P(\text{"he likes to drink coffee"}) \\ &= P(\text{he}) + P(\text{likes} | \text{he}) + P(\text{to} | \text{he likes}) \\ &\quad + P(\text{drink} | \text{likes to}) + P(\text{coffee} | \text{to drink}) \end{aligned}$$

# Ngram Language Model

- Count and divide

$$P_{ML}(x_i \mid x_{i-n+1}, \dots, x_{i-1}) := \frac{c(x_{i-n+1}, \dots, x_i)}{c(x_{i-n+1}, \dots, x_{i-1})}$$

- Add smoothing to deal with zero counts

$$\begin{aligned} P(x_i \mid x_{i-n+1}, \dots, x_{i-1}) &= \lambda P_{ML}(x_i \mid x_{i-n+1}, \dots, x_{i-1}) \\ &\quad + (1 - \lambda) P(x_i \mid x_{1-n+2}, \dots, x_{i-1}) \end{aligned}$$

# Evaluating Language Model

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word (Cross) Entropy:**

$$H(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} -\log_2 P(E)$$

- **Perplexity:**

$$ppl(\mathcal{E}_{test}) = 2^{H(\mathcal{E}_{test})} = e^{-WLL(\mathcal{E}_{test})}$$

# Perplexity

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Normalized by  
number of words

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

Inverse probability  
of test set

# Perplexity

- Maximizing probability = Minimizing perplexity
- What is perplexity in Deep Learning models?

$$\text{NLLoss} = \ln \mathcal{L}(\theta) = - \sum_{i=1}^n y_i \ln(\hat{y})$$

Lower is better!!!

$$\text{CE} = \mathcal{L}(\theta) = -\frac{1}{n} \sum_{i=1}^n y_i \ln(\hat{y})$$

# Generating text with Ngram LM

**While** not </s>:

**calculate** probability of possible next words

**choose** a word from the top N most probable

<context> = "<s> he likes to"

$P(\text{drink} \mid \text{context}) > P(\text{coffee} \mid \text{context})$

# Limitations of Ngram Language Models

- Storage and retrieving ngrams probabilities
- Sparsity and smoothing hacks
- Context windows limits long-distance dependencies
- Frequencies says little about semantics

# The Shannon Game

- Life is like a box of \_\_\_\_\_
- Life is too short to miss out on the beautiful things like a double \_\_\_\_\_
- Science is organized knowledge, wisdom is organized \_\_\_\_\_

# The Shannon Game

- Life is like a box of **chocolate**
- Life is too short to miss out on the beautiful things like a double **cheeseburger**
- Science is organized knowledge, wisdom is organized **life**

# Language Model Evaluation

- A language model that can ***predict the right words***, i.e. ***assign a higher probability to words that occurs*** is a better model
- Traditionally, LM is evaluated on **perplexity**
- Perplexity is the ***inverse probability of the test set***, normalized by the number of words

# RNN vs N-gram Language Model

N-gram  
model

RNN with  
increased  
complexity

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
<b>Ours small</b> (LSTM-2048)	43.9
<b>Ours large</b> (2-layer LSTM-2048)	39.8

From <https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/>

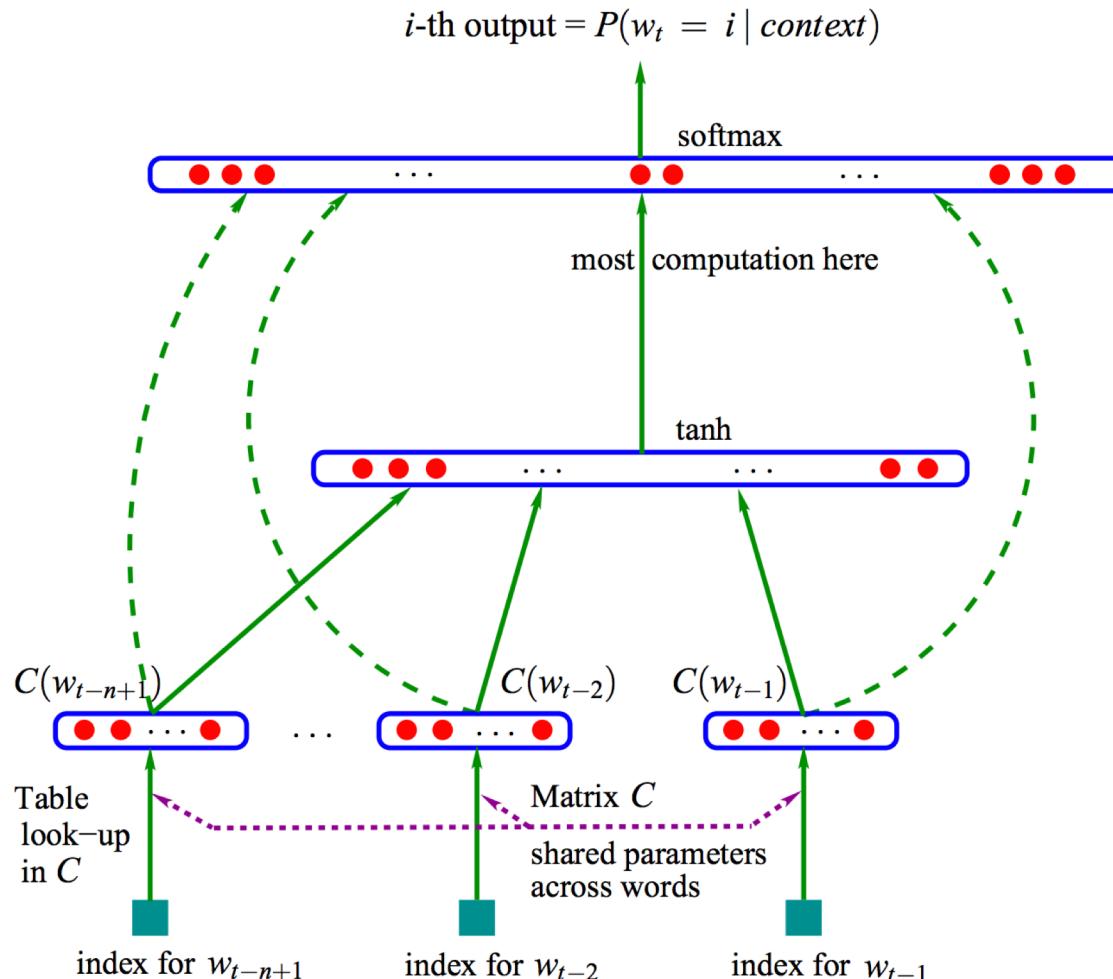


# Neural Language Modelling

# Deep Language Models

- Calculate weights/features of context
- Based on the weights, compute probabilities
- Optimize weights using gradient descent to minimize errors on probabilities computation

# A Neural Probabilistic Language Model



(Bengio et al. 2004)

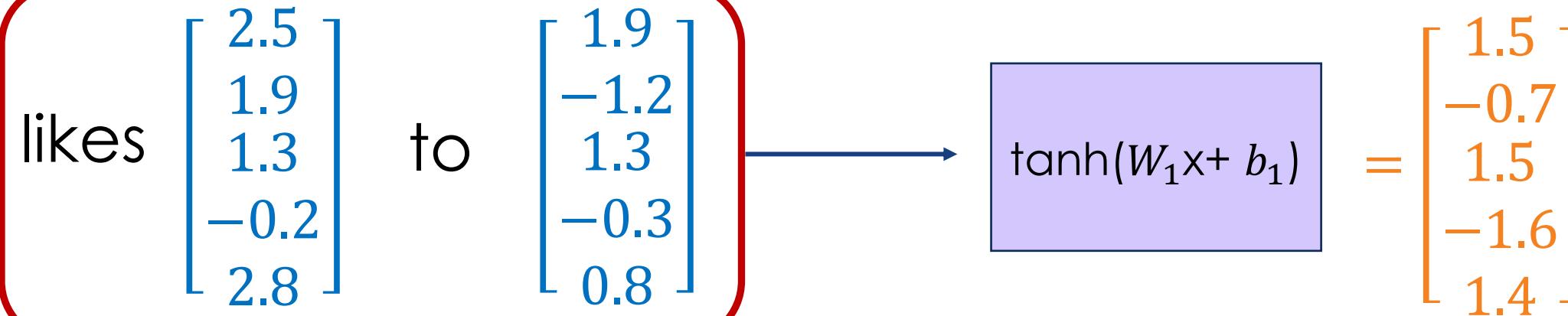
Figure 1: Neural architecture:  $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$  where  $g$  is the neural network and  $C(i)$  is the  $i$ -th word feature vector.

# Neural Language Model

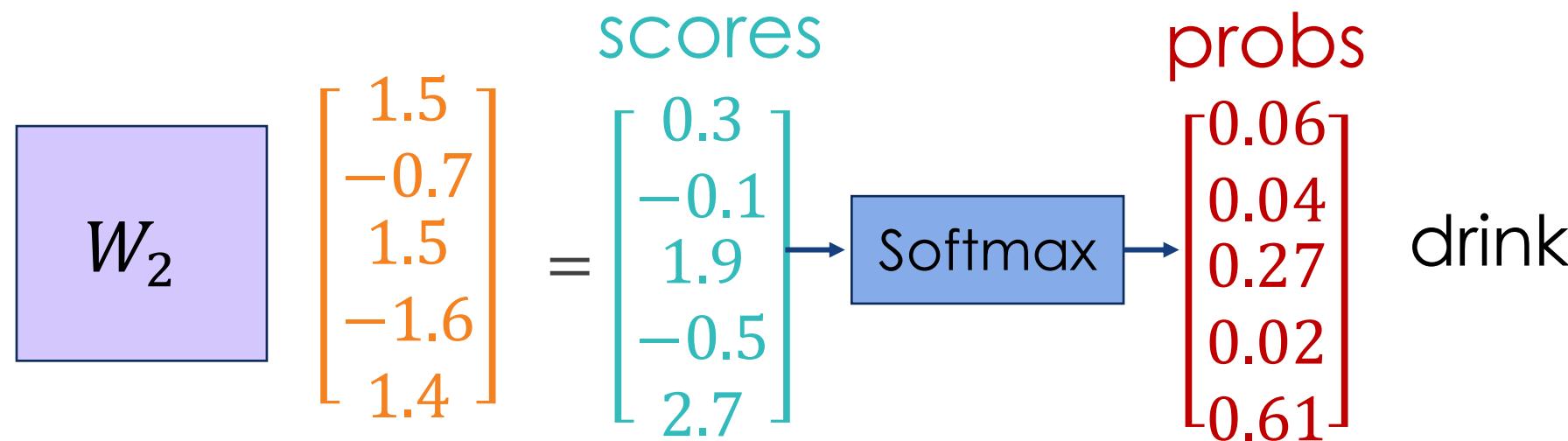
likes  $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$  to  $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$   $\longrightarrow$   $\tanh(W_1x + b_1)$   $= \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

$W_2$   $\begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix} = \begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix} \xrightarrow{\text{Softmax}} \text{probs} \begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}$  drink

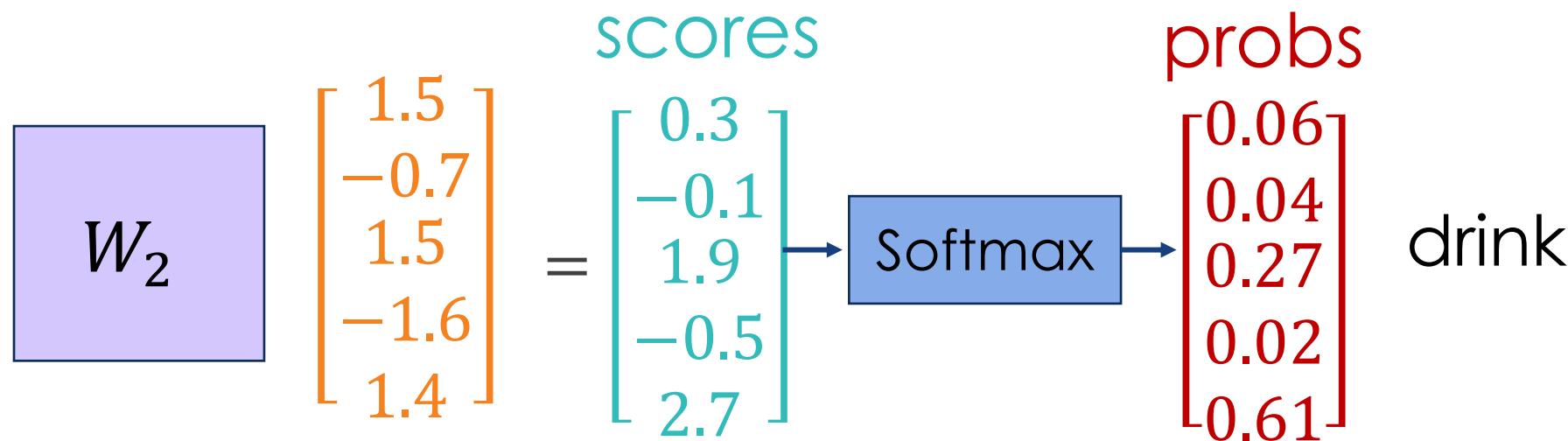
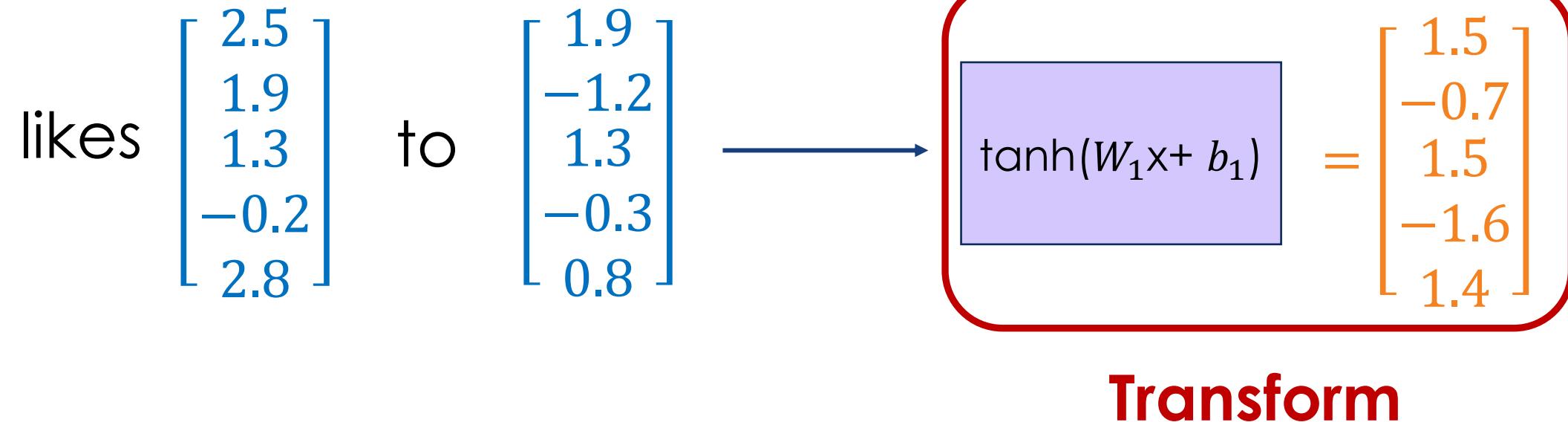
# Neural Language Model



## Lookup function



# Neural Language Model

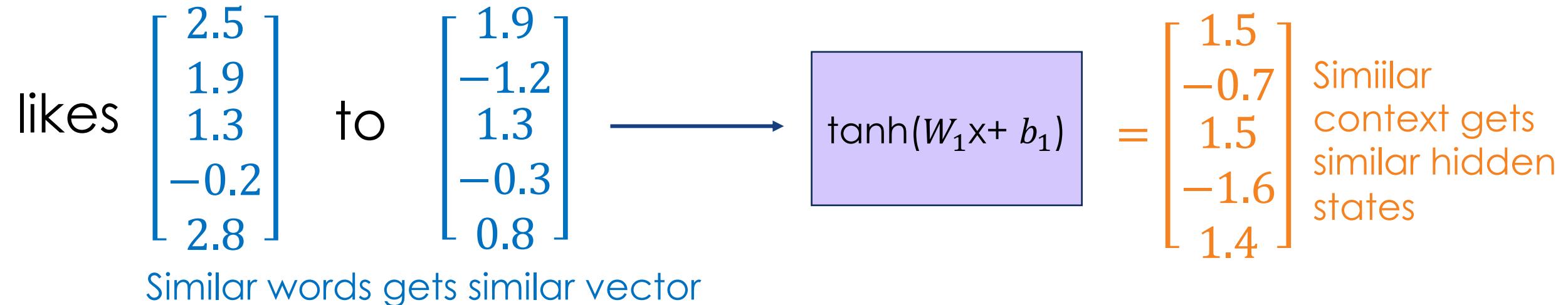


# Neural Language Model

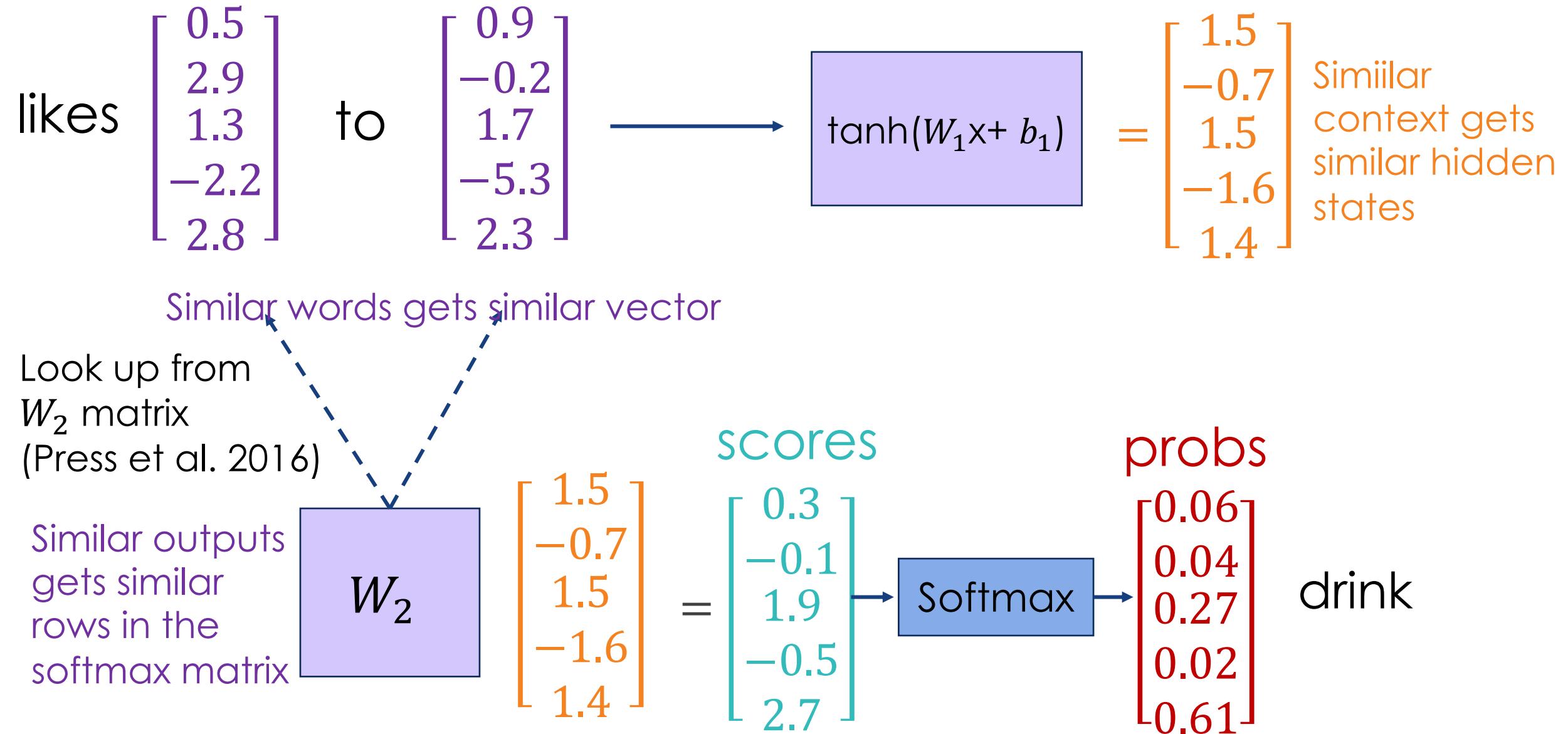
likes  $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$  to  $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$   $\longrightarrow$   $\tanh(W_1x + b_1)$   $= \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

Predict  $W_2$   $\begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$  = scores  $\begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix}$   $\rightarrow$  Softmax  $\rightarrow$  probs  $\begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}$  drink

# Neural Language Model



# Neural Language Model



# Loss Function for Language Models

Loss function on step  $t$  is usual cross-entropy between our predicted probability distribution  $\hat{y}^{(t)}$ , and the true next word  $y^{(t)} = x^{(t+1)}$ :

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- Calculate weights/features of context
- Based on the weights, compute probabilities
- Optimize weights using gradient descent to minimize errors on probabilities computation
- **When do we stop?** If we have softmax as the last layer activation, cost function is most probably?

# Cost Function for Neural LM

- Negative Log Loss Function (`torch.nn.NLLoss`)
- Maximum Likelihood Estimation (MLE) ==  
Minimize the NLLoss of all sentences
- i.e. Find the parameters that make the sentences  
in the training data most likely

- **Shuffle the input sentences**
  - Imagine seeing “*Justin Bieber is the best singer, baby, baby, baby, oh*” 100x at the start of the corpus...
- **Early stopping** based on some validation set
- **Dropout** during training

- **Shuffle the input sentences**
  - Imagine seeing “*Justin Bieber is the best singer, baby, baby, baby, oh*” 100x at the start of the corpus...
- **Early stopping** based on some validation set
- **Dropout** during training
- **Mini-batching** makes training much faster

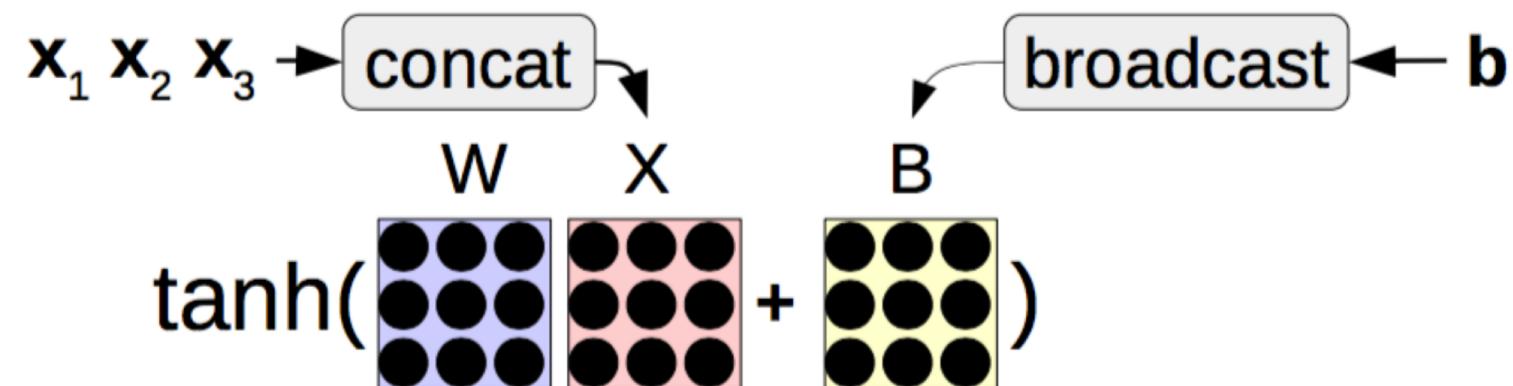
# Mini-batching

## Operations w/o Minibatching

$$\tanh(\begin{matrix} W & x_1 & b \end{matrix} + \begin{matrix} W & x_2 & b \end{matrix} + \begin{matrix} W & x_3 & b \end{matrix})$$

Three separate operations are shown, each consisting of a weight matrix  $W$  (purple), an input vector  $x$  (pink), and a bias vector  $b$  (yellow). The vectors  $x_1, x_2, x_3$  are stacked vertically.

## Operations with Minibatching



# Environment Setup

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration:

<https://pytorch.org/get-started/locally/>

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download <http://bit.ly/ANLP-Session5NgramLM>

Import the .ipynb to the Jupyter Notebook



# Summary

- **Language Modelling**
  - LM is predicting the next word given history
  - Negative Log Loss as cost function for simple LM
  - Shuffle data, stop early on validation set, always dropout

# Hands-on PyTorch Checklist

- How to use `torch.utils.data.DataLoader`
- How to build a classifier using multi-layered perceptron